OTTO VON GUERICKE UNIVERSITÄT MAGDEBURG | INF | FAKULTÄT FÜR INFORMATIK

Magdeburg, den 13.09.2024

Wiederholungsklausur zur Vorlesung

Einführung in die Informatik für Ingenieure (Winter 23/24)

Name: _____

Vorname: _____

Matrikelnummer: _____

Bitte beachte die folgenden Hinweise:

- Du hast **120 Minuten** zum Schreiben dieser Klausur.
- Bitte lege deinen Ausweis und deinen Studierendenausweis für die Anwesenheitskontrolle bereit.
- Die Klausur besteht aus 14 Seiten. Bitte prüfe zu Beginn, ob du alle Seiten bekommen hast.
- Bevor du beginnst, eine Aufgabe zu bearbeiten, lies dir bitte die Aufgabenstellung genau durch! Schreibe deine Antworten in die dafür vorgesehen Kästchen oder direkt in die Aufgabenstellung. Sollte der Platz hierfür nicht ausreichen, kannst du die Rückseite des **vorherigen** Blattes zum Schreiben verwenden. Du darfst kein eigenes Papier verwenden.
- Streiche falsche Lösungen deutlich durch.
- Benutze keine Bleistifte und weder Stifte mit roter oder grüner Tinte.
- Es sind **keine** Hilfsmittel außer eines Stifts zugelassen.
- Solltest du zu einer Aufgabe mehr als eine Lösung abgeben, wird keine der Lösungen gewertet.
- Wenn du während der Klausur fragen haben solltest, melde dich einfach. Wir kommen dann zu dir und versuchen zu helfen.
- Wenn du den Raum verlassen musst, müssen wir das protokollieren. Es kann immer höchstens eine Person zur gleichen Zeit den Raum verlassen. Wenn du den Raum verlassen möchtest, melde dich bitte, damit wir dir Bescheid geben können, sobald du gehen kannst.
- **Täuschungsversuche führen zum Nichtbestehen der Klausur.**

Viel Erfolg! 😀

Aufgabe	1	2	3	4	5	6	7	8	$\sum$
maximale Punkte	11	4	6	8	10	9	7	5	60
erreichte Punkte									

OTTO VON GUERICKE UNIVERSITÄT MAGDEBURG | INF | FAKULTÄT FÜR INFORMATIK

Magdeburg, den 13.09.2024

Aufgabe 1 *: Grundlegende Algorithmen*

a *) Der folgende Code implementiert eine rekursive Version der binären Suche:*

```
1  def find(a: list[int], x: int) -> int | None:
2      def _find(a: list[int], x: int, l: int, r: int) -> int | None:
3          if l > r:
4              return None
5          m: int = (l+r) // 2
6          if a[m] == x:
7              return m
8
9          if x < a[m]:
10             return _find(a, x, l, m - 1)
11         else:
12             return _find(a, x, m + 1, r)
13
14     return _find(a, x, 0, len(a)-1)
```

- *Welche Bedeutung hat der Rückgabewert von* `find`*?*
- *Definiere und implementiere eine iterative Funktion* `ifind` *für die binäre Suche. Deine Implementierung darf keine Rekursion verwenden und keine Bibliotheken oder vordefinierten Funtionen benutzen.*
- *Angenommen, du dürftest nur 11 Vergleiche machen: Wie groß könnte ein Array maximal sein, damit du garantiert ein richtiges Ergebnis bekommst? (**Hinweis:** Um die Aufgabe zu vereinfachen, brauchst du nur die Vergleiche in Zeile 9 zählen und kannst alle anderen Vergleiche ignorieren.)*

[5 points]

Magdeburg, den 13.09.2024

b) *Gegeben seien die folgenden Funktionen. Erkläre, was die jeweiligen Funktionen berechnen und was deren jeweilige Komplexitätsklasse im* schlimmsten Fall *ist (Groß-O-Notation!).*

```
1  def a(n: int) -> int:
2      if n <= 1:
3          return 1
4      else:
5          return n * a(n-1)
6
7
8  def b(n: int) -> int:
9      if n <= 0:
10         return 0
11     else:
12         return 1 + b(n//2)
13
14 def c(n: int) -> int:
15     if n <= 0:
16         return 0
17     elif n <= 2:
18         return 1
19     else:
20         return c(n-2) + c(n-1)
```

[6 points]

/ 11

OTTO VON GUERICKE UNIVERSITÄT MAGDEBURG | INF | FAKULTÄT FÜR INFORMATIK

Magdeburg, den 13.09.2024

Aufgabe 2 : *Objektorientierte Programmierung*

a) *Gegeben sei die folgende Klassenhierarchie, was ist die Ausgabe, die von den Zeilen 30, 33, 34, and 37 erzeugt wird?*

```
1  class Pokemon:
2      def attack(self, s: str) -> None:
3          raise NotImplementedError
4
5      def move(self, target: str) -> None:
6          print(f"Teleporting to {target}.")
7
8  class Eevee(Pokemon):
9      def attack(self, s: str = "") -> None:
10         print("Tackle!")
11
12     def move(self, target: str) -> None:
13         print(f"Run away from {target}.")
14
15 class Vaporeon(Eevee):
16     def attack(self, target: str) -> None:
17         super().move(target)
18         print("Water Gun!")
19         self.attack(target)
20
21 class Magikarp(Pokemon):
22     def attack(self, s: str) -> None:
23         super().attack(s)
24
25     def move(self, target: str) -> None:
26         print(f"Flop around.")
27
28 if __name__ == "__main__":
29     garados: Magikarp = Magikarp()
30     garados.move("Kanto")
31
32     jolteon: Eevee = Eevee()
33     jolteon.attack()
34     jolteon.move("Alola")
35
36     flareon: Pokemon = Vaporeon()
37     flareon.attack("Pikachu")
```

[4 points]

	Output
Zeile 30:	
Zeile 33:	
Zeile 34:	
Zeile 37:	

/ 4

Magdeburg, den 13.09.2024

Aufgabe 3 *: Listen*

a *)*

```
1  class DoubleLinkedNode:
2      def __init__(self, item=None, prev=None, next=None):
3          self._item = item
4          self._prev = prev
5          self._next = next
6
7
8  class DoubleLinkedList:
9      def __init__(self):
10         self._head = DoubleLinkedNode()                    # an empty DLL always
11         self._tail = DoubleLinkedNode(prev=self._head)     # contains at least
12         self._head._next = self._tail                      # two empty nodes
13
14     def insert_before(self, node: DoubleLinkedNode):
15         pass
16
17     def remove(self, node: DoubleLinkedNode):
18         pass
```

Implementiere die beiden fehlenden Funktionen `insert_before` *und* `remove`*. Du darfst davon ausgehen, dass alle Argumente ausreichend instantiiert wurden (d.h. nicht* `None`*). Außerdem darfst du im Fall von* `remove` *davon ausgehen, dass der übergebene Knoten auch wirklich Teil der Liste ist. Du kannst alle Attribute und Methoden der Klasse* `DoubleLinkedNode` *direkt benutzen, ohne Getter oder Setter Methoden benutzen zu müssen. [4 points]*

Magdeburg, den 13.09.2024

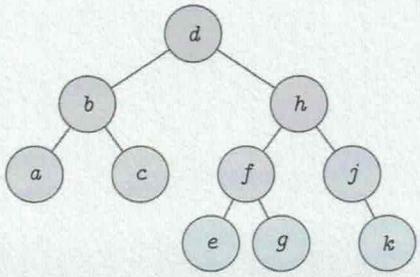
b *) Die Implementierung der doppelt verketteten Liste aus der Vorlesung und die von der vorherigen Aufgabe benutzen jeweils Dummy-Knoten für den* `_head` *und den* `_tail` *der Liste. Warum ist das so? Was würde sich ändern, wenn wir das erste bzw. letzte Element direkt referenzieren würden? [2 points]*

/ 6

Magdeburg, den 13.09.2024

Aufgabe 4 *: Bäume*

a *) Gegeben sei der folgende binäre Suchbaum:*



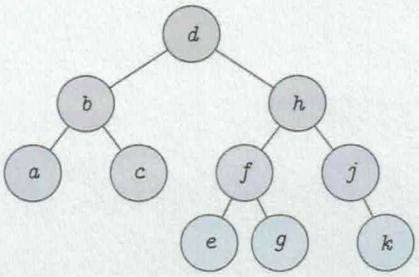
- *Lösche – in genau dieser Reihenfolge – die Knoten* `a`*,* `j` *und* `d`*. Zeichne den Baum nach jedem Löschvorgang.*

[4 points]

Magdeburg, den 13.09.2024

b *) Gegeben sei der folgende binäre Baum. Was ist das Ergebnis eines*

- *Preorder-Traversals,*
- *Inorder-Traversals,*
- *Postorder-Traversals,*
- *Pevelorder-Traversals?*



preorder traversal:	
inorder traversal:	
postorder traversal:	
levelorder traversal:	

[4 points]

/ 8

OTTO VON GUERICKE UNIVERSITÄT MAGDEBURG | INF | FAKULTÄT FÜR INFORMATIK

Magdeburg, den 13.09.2024

Aufgabe 5 *: Balancierte Bäume*

a *)*

- *Erkläre* **kurz**, *was ein AVL-Baum ist.*
- *Füge die folgende Sequenz von Zahlen in einen AVL-Baum ein. Skizziere den Baum nach jedem Einfügen und vor/nach jedem Umordnen.*

`[5, 2, 1, 6, 3, 4, 7, 0]`

[5 points]

Magdeburg, den 13.09.2024

b)

- *Erkläre* **kurz**, *was ein 2-3-4 Baum ist.*
- *Füge die folgende Sequenz von Zahlen in einen 2-3-4 Baum ein. Skizziere den Baum nach jedem Einfügen und vor/nach jedem Umordnen.*

`[50, 20, 60, 10, 8, 15, 32, 26, 48, 42]`

[5 points]

/ 10

Magdeburg, den 13.09.2024

Aufgabe 6 *: Hashs*

a *) Füge die folgenden Zahlen in eine Hashtabelle $\mathcal{H}$ ein.*

`[15, 9, 1, 10, 8, 3]`

Die Hashtabelle hat die Größe $m = 7$. Benutze die beiden folgenden Hashfunktionen:

$$h_1(x) = 2x - 1$$

$$h_2(x) = 3x + 1.$$

x	1	3	8	9	10	15
$h_1(x)$						
$h_2(x)$						

---

1. *Benutze offene Adressierung mit linearem Sondieren zur Kollisionsbehandlung. Benutze für diese Aufgabe h_1 als Hashfunktion! Zähle außerdem die Anzahl der Kollissionen.*

i	0	1	2	3	4	5	6
$\mathcal{H}[i]$							

2. *Benutze offene Adressierung mit Double-Hashing zur Kollisionsbehandlung. Benutze für diese Aufgabe h_1 und h_2 als Hashfunktionen! Zähle außerdem die Anzahl der Kollissionen.*

i	0	1	2	3	4	5	6
$\mathcal{H}[i]$							

3. *Welchen Vorteil hat Double-Hashing normalerweise gegenüber linearem Sondieren?*
4. *Um Kollissionen beim Einfügen zu vermeiden könnten wir den Wert der Hashfunktion auch zufällig bestimmen. Warum ist das eine schlechte Idee?*

[9 points]

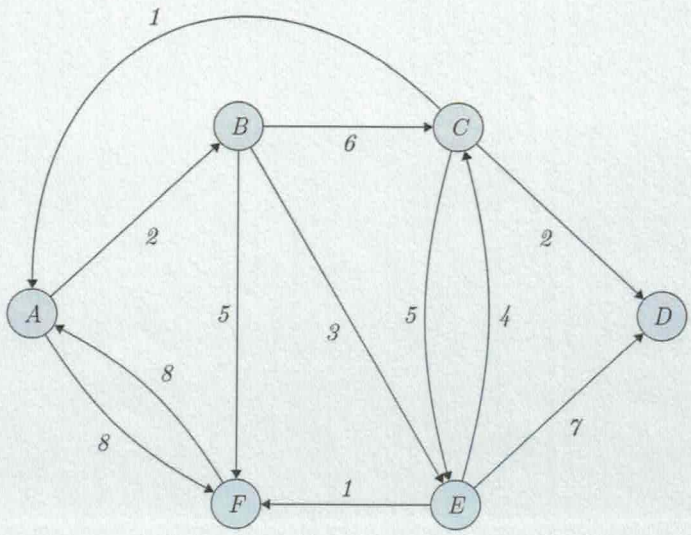
/ 9

OTTO VON GUERICKE UNIVERSITÄT MAGDEBURG | INF | FAKULTÄT FÜR INFORMATIK

Magdeburg, den 13.09.2024

Aufgabe 7 *: Graphen*

a *) Wende Dijkstras Algorithmus auf den folgenden Graphen an. Beginne dafür mit dem Knoten* **A**.

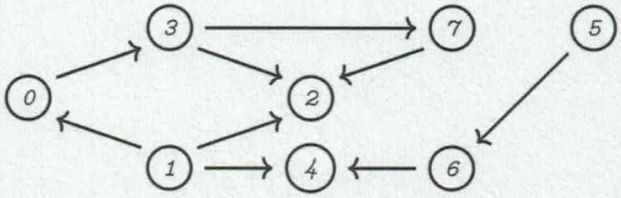


	A	B	C	D	E	F
Vorgänger	-					
Abstand zu A	0					

[4 points]

Magdeburg, den 13.09.2024

b *) Gegeben sei der folgende Graph:*



Berechne eine mögliche topologische Sortierung dieses Graphen. Falls du jemals die Wahl zwischen mehr als einem Knoten hast, wähle zum weitermachen immer den Knoten aus, der die kleinste Zahl enthält. [3 points]

/ 7

Magdeburg, den 13.09.2024

Aufgabe 8 *: Wahr oder Falsch*

a *) Entscheide für die folgenden Aussagen, ob sie wahr oder falsch sind. Du musst deine Antwort nicht begründen. Jede richtige Antwort gibt 0.5 Punkte, jede falsche Antwort gibt 0.5 Punkte Abzug. Wenn du eine Frage nicht beantwortest kriegst du weder Punkte, noch werden dir Punkte abgezogen. Bei dieser Aufgabe kriegst du mindestens 0 Punkte.*

1. $log_3 2^n \in \mathcal{O}(n)$
2. *Im besten Fall kann Bubble Sort schneller sein als Insertion Sort.*
3. *Der schnellste Weg einen Heap aus einem Array aufzubauen liegt in* $\mathcal{O}(n)$.
4. *Das* B *in B-Baum steht für* binär.
5. *Merge Sort kann als externes Sortierverfahren genutzt werden.*
6. *Pythonprogramme werden vor der Ausführung kompiliert.*
7. *Klassen können in Python von mehr als einer Basisklasse erben.*
8. *Wenn man in Python einem Attribut den Präfix* `__` *gibt, heißt das, dass man von außerhalb dieser Klasse nicht mehr auf das Attribut zugreifen kann.*
9. *In einem binären Min-Heap befindet sich das größte Element in einem der Blätter.*
10. *Zum Lösen des Rucksackproblems mit dynamischer Programmierung braucht man* $\mathcal{O}(n^C)$ *Schritte.*

	true	*false*
1	☐	☐
2	☐	☐
3	☐	☐
4	☐	☐
5	☐	☐
6	☐	☐
7	☐	☐
8	☐	☐
9	☐	☐
10	☐	☐

[5 points]

/ 5